Python.org Website Documentation

Release 1.0

Python Software Foundation

January 21, 2017

Contents

Documentation for the code behind python.org.

CHAPTER 1

General information

Source code https://github.com/python/pythondotorg

Issue tracker https://github.com/python/pythondotorg/issues

Mailing list pydotorg-www

IRC `#pydotorg` on Freenode

Staging site https://staging.python.org/ (`master` branch)

Production configuration https://github.com/python/psf-salt

Travis

License Apache License

Contents:

1.1 Installing

Here are two ways to hack on python.org:

1. *Easy setup using Vagrant*
2. *Manual setup*

1.1.1 Easy setup using Vagrant

```
$ vagrant up
$ vagrant ssh
# go to pythondotorg/ directory and activate virtualenv, then run
$ ./manage.py runserver 0.0.0.0:8000
# on your local shell
$ google-chrome http://localhost:8001/
```

The box will be provisioned by Ansible 1.9.3 with Python 3.4, a virtualenv set up with requirements installed, and a database ready to use.

The box also creates a superuser with username `cbiggles` for you. However, you will need to set a password before using it:

```
$ vagrant ssh
$ cd pythondotorg
$ . venv/bin/activate
$ ./manage.py changepassword cbiggles
```

Note: You will also need to run `./manage.py load_dev_fixtures` to load all fixture files. This will download an approximately 11 MB gzipped set of fixtures that are sanitized of sensitive user data and then loaded into your local database.

1.1.2 Manual setup

First, clone the repository:

```
$ git clone git://github.com/python/pythondotorg.git
```

Then create a virtual environment:

```
$ python3.4 -m venv venv
```

And then you'll need to install dependencies:

```
$ pip install -r dev-requirements.txt
```

pythondotorg will look for a PostgreSQL database named `pythondotorg` by default. Run the following command to create a new database:

```
$ createdb pythondotorg -E utf-8 -l en_US.UTF-8
```

To change database configuration, you can add the following setting to `pydotorg/settings/local.py` (or you can use the `DATABASE_URL` environment variable):

```
DATABASES = {
    'default': dj_database_url.parse('postgres:///your_database_name')
}
```

Now it's time to run migrations:

```
$ ./manage.py migrate
```

To compile and compress static media, you will need *compass* and *yui-compressor*:

```
$ gem install bundler
$ bundle install
```

Note: To install *yui-compressor*, use your OS's package manager or download it directly then add the executable to your `PATH`.

To load all fixture files:

```
$ ./manage.py load_dev_fixtures
```

Note: This will download an approximately 11 MB gzipped set of fixtures that are sanitized of sensitive user data and then loaded into your local database.

Finally, start the development server:

```
$ ./manage.py runserver
```

1.1.3 Optional: Install Elasticsearch

The search feature in Python.org uses Elasticsearch engine. If you want to test out this feature, you will need to install Elasticsearch.

Once you have it installed, update the URL value of `HAYSTACK_CONNECTIONS` settings in `pydotorg/settings/local.py` to your local ElasticSearch server.

1.1.4 Generating CSS files automatically

Due to performance issues of django-pipeline, we are using a dummy compiler `pydotorg.compilers.DummySASSCompiler` in development mode. To generate CSS files, use `sass` itself in a separate terminal window:

```
$ cd static
$ sass --compass --scss -I $(dirname $(dirname $(gem which susy))) --trace --watch sass/style.scss:sa
```

1.1.5 Running tests

To run the test suite:

```
$ ./manage.py test
```

To generate coverage report:

```
$ coverage run manage.py test
$ coverage report
```

Generate an HTML report with `coverage html` if you like.

1.1.6 Useful commands

- Create a super user (for a new DB):

```
$ ./manage.py createsuperuser
```

- Want to save some data from your DB before nuking it, and then load it back in?:

```
$ ./manage.py dumpdata --format=json --indent=4 $APPNAME > fixtures/$APPNAME.json
```

1.1.7 Troubleshooting

If you hit an error getting this repo setup, file a pull request with helpful information so others don't have similar problems.

Freetype not found on OSX

```
_imagingft.c:60:10: fatal error: 'freetype/fterrors.h' file not found
#include <freetype/fterrors.h>
         ^
1 error generated.
error: command 'clang' failed with exit status 1
```

If you've installed *freetype* (`brew install freetype`), you may need to symlink version 2 into location for version 1 as mentioned by this Stack Overflow question.

Freetype 2.5.3 is known to work with this repository:

```
$ ln -s /usr/local/include/freetype2 /usr/local/include/freetype
```

1.1.8 Building documentation

If you want to install the default Read the Docs theme, you can do:

```
$ pip install -r docs-requirements.txt
```

To build this documentation locally:

```
$ make -C docs/ htmlview
```

If you don't want to open the browser automatically, you can do:

```
$ make -C docs/ html
```

1.2 Contributing

1.2.1 Bugs

All source and content bugs for python.org should be filed as issues on the GitHub. Please be sure to look at the existing issues to see if someone has already submitted it.

1.2.2 Code

The source for python.org is open and licensed under the Apache 2 license. To contribute to either the code or documentation please fork the pythondotorg repository and submit a pull request.

See Installing for setup your development environment.

While all contributions are welcome and valuable it will be easier for the maintainers if you follow good coding practices such as:

- Write readable code following **PEP 8** guidelines
- Write tests for the code you are adding or changing
- Submit focused pull requests that address a single smaller issue whenever possible vs a large pull request that touches many parts of the system
- Submit descriptive information with your pull request as to the reason and methods behind your change

1.3 Administration

1.3.1 Navigation

Navigation on the site is managed by the Sitetree application. The hierarchy should be fairly obvious. The biggest gotcha is when defining the URLs.

Many URLs are defined using Django's URL system however many are also simply defined as relative paths. When editing a particular item in the Sitetree in the *Additional Settings* fieldset there is an option named *URL as pattern*. If this option is checked the URL pattern is checked against the URLs defined by the Django applications. If it is left unchecked relative and absolute URLs can be entered.

1.3.2 Supernavs

The concept of a *Supernav* is used heavily on the site. These are the larger bits of text and markup included in the main navigation drop downs. These are implemented as specially named *Boxes*. They are by convention named `supernav-*`, for example `templates/downloads/supernav.html`.

Here is an example of what that looks like on the site.

The sub-nav items on the left are simply nested *Navigation* links in SiteTree relative to the 'Download' item in the tree. The larger *Download for Mac OSX* box however is what we refer to as the *supernav*.

Most supernavs are updated automatically based on the underlying Django application content using signals. By convention the application will have a template named `supernav.html`. For example, upon saving any published *Release* a Django signal is fired to update the `supernav-python-downloads` box with the most current Python2 and Python3 releases. In this case the markup is structured in a way to allow for the automatic OS detection Javascript to show the user the appropriate download links for the OS they are browsing with.

1.3.3 Pages

Pages are individual entire pages of markup content. They are require `Title`, `Path`, and `Content` to be acceptable in the system. Note that Pages are implemented using a fall through system of URL routing so a user cannot override an existing defined Django URL on accident with a Page.

Fields and Descriptions

Title Title of the Page. Will also be used as the `<title></title>` attribute in the markup.

Keywords HTML META keywords for search engines

Description HTML META description for search engines

Path Relative URL path where the page will reside, excluding the initial slash. *Example*: `about/psf/somepage/`

Content The actual content of the page.

Markup Type Type of markup contained in the *Content* field. Options are: HTML, plain text, ReStructured Text, and Markdown

Is Published Controls whether or not the page is visible on the site.

Template Name By default Pages use the template `templates/pages/default.html` to use a different template enter the template path here.

Note: Pages are automatically purge from Fastly.com upon save.

1.3.4 Boxes

Boxes are re-usable bits of HTML markup that are used throughout the site. Things like sidebar info and specific areas of areas of pages with a richer design (i.e. landing pages) that would be cumbersome to edit as one large content textarea.

Note: There are *special boxes* that are automatically rebuilt using templates, see *Supernavs*.

1.3.5 Downloads and Releases

The `downloads` app stores all of the structured data regarding Python releases. Each `Release` object has associated `ReleaseFile` objects that contain information on the various download formats Python.org supports.

If the version you are creating should be considered the "latest" release for the major version in question (Python 2.x.x, 3.x.x, etc) then check the 'Is this the latest release' checkbox. When the `Release` is saved, the previous version will be automatically demoted for you and the new version will be used prominently on the site. For example the download buttons and supernav links.

Note: If you make a mistake here, no worries you can just check the box and save on **ANY** version and promote it to being the latest release.

To create a release you simply need to fill in the appropriate information. Currently if a `Release` has an associated `Release Page` the system redirects to that to accommodate legacy content, but if the `Content` field is filled they are taken to the Release Detail page which shows the content and lists all of the associated downloadable files.

Release Files have a checkbox named 'Download button' that determines which binary/source package download link to display for a given OS. This information is used by the OS detection JS on the site so pick the package in most widespread use. On Source distributions be sure to check the 'Download button' for the .tgz version for widest compatibility.

1.3.6 Jobs

The jobs application is using to display Python jobs on the site. The data items should be fairly self explanatory. There are a couple of things to keep in mind. Logged in users of the site can submit jobs for review.

Status Jobs enter the system in 'review' status after the submitter has entered them. Only jobs in the 'approved' state are displayed on the site.

Featured Featured jobs are displayed more prominently on the landing page.

Comments Users who have submitted a job and admin reviewers can make comments on a job. Emails will be sent to the other party in order to foster communication about job description and data.

1.3.7 Sponsors

The Sponsors app is a place to store PSF Sponsors. Sponsors have to be associated to a Company model from the companies app. If they are marked as *is_published* they will be shown on the main sponsor page which is located at /sponsors/.

If a Sponsor is marked as *featured* they will be included in the sponsor rotation on the main PSF landing page. In the fourth "Sponsors" column.

1.3.8 Events

TODO

1.3.9 Companies

TODO

1.3.10 Success Stories

TODO

1.4 PEP Page Generation

1.4.1 Process Overview

We are generating the PEP pages by lightly parsing the HTML output from the PEP Repository and then cleaning up some post-parsing formatting.

The PEP Page Generation process is as follows:

1. Clone the PEP Repository, if you have not already done so:

```
$ git clone https://github.com/python/peps.git
```

2. From the cloned PEP Repository, run:

```
$ make -j
```

3. Set `PEP_REPO_PATH` in `pydotorg/settings/local.py` to the location of the cloned PEP Repository
4. Run in your `pythondotorg` repository:

```
$ ./manage.py generate_pep_pages
```

This process runs periodically via cron to keep the PEP pages up to date.

1.4.2 Management Commands

generate_pep_pages

This Django management command generates `pages.Page` objects from the output of the existing PEP repository generation process. You run it like:

```
$ ./manage.py generate_pep_pages
```

To get verbose output run it like:

```
$ ./manage.py generate_pep_pages --verbosity=2
```

It uses the conversion code in the `peps.converters` module in an attempt to normalize the formatting for display purposes.

dump_pep_pages

This simply dumps our PEP related pages as JSON. The `dumpdata` content is written to `stdout` just like a normal `dumpdata` command.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

P